JP11110256

Publication Title:

DEVICE AND METHOD FOR DEBUGGING PROGRAM, AND COMPUTER READABLE RECORDING MEDIUM RECORDED WITH THE METHOD FOR THE SAME

Abstract:

Abstract of JP 11110256

(A) Translate this text PROBLEM TO BE SOLVED: To debug a program by generating an object file so that the debugging object and the final object are managed with the same execution object a mem ory of target machine side is not burdened in debugging. SOLUTION: An object file 24 generated by a compiler 21 based on a source file 23 is divided into a debug script file activated in debugging, an execution object file which does not include a debug script, and a symbol table of debug script activation information relating the debug script with a position of a source line. In downloading the execution object file in a target machine 11, executing this file and debugging it, when an execution of the program reaches the debug script part, a debugger 22 executes a debug script file on a host machine on the basis of debug script activation information and executes a debug.

---

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平11－110256

(43)公開日　平成11年(1999)４月23日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　11/28	３０５	Ｃ０６Ｆ　11/28	３０５Ｚ
9/45		9/44	３２２Ｚ

審査請求　未請求　請求項の数27　ＯＬ　（全 11 頁）

(21)出願番号　特願平9－272792

(22)出願日　平成９年(1997)10月６日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　宮本　出
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内

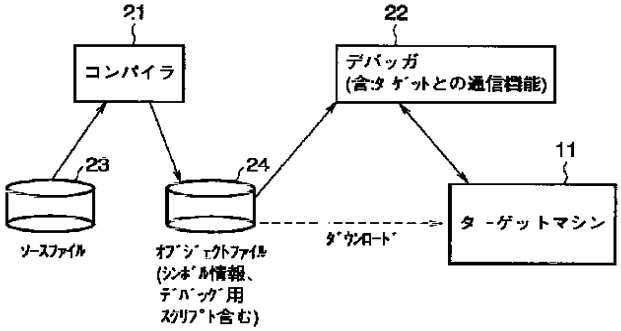
(74)代理人　弁理士　鈴江　武彦　（外６名）

(54)【発明の名称】　プログラムデバッグ装置、プログラムデバッグ方法及びその方法を記録したコンピュータ読取り可能な記録媒体

(57)【要約】

【課題】本発明はデバッグ時物件と最終物件と同じ実行オブジェクトで管理でき、デバッグ処理時にターゲットマシン側のメモリ負担がないようにオブジェクトファイルを生成しデバッグすることを課題とする。

【解決手段】ソースファイル２３をもとにコンパイラ２１により生成したオブジェクトファイル２４は、デバッグ時起動のデバッグスクリプトファイルと、デバッグスクリプトを含まない実行オブジェクトファイルと、デバッグスクリプトとソース行の位置を対応付けるデバッグスクリプト起動情報のシンボルテーブルとに分けられる。ターゲットマシン１１に実行オブジェクトファイルをダウンロードし、このファイルを実行しデバッグする際、デバッグスクリプト部分にプログラムの実行が到達した時、デバッグスクリプト起動情報に基きデバッガ２２がホストマシン１０上のデバッグスクリプトファイルを実行しデバッグを行う。



【特許請求の範囲】
【請求項１】　プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルするコンパイラと、
デバッグ時に上記デバッグスクリプトを有効にするか否かを指定するデバッグ指定手段と、
このデバッグ指定手段により上記デバッグスクリプトを有効にしたとき、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定する判定手段とを具備し、
上記判定手段により上記デバッグスクリプトにプログラムの制御が到達したことを判定したときに、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とするプログラムデバッグ装置。
【請求項２】　上記コンパイラは、上記ソースファイルをコンパイルする際に、所定の記述により上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項１記載のプログラムデバッグ装置。
【請求項３】　上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記コンパイラは、上記ソースファイルをコンパイルする際に上記識別子にて上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項１記載のプログラムデバッグ装置。
【請求項４】　上記デバッグ指定手段は、上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、
上記判定手段は、上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項１乃至３のいずれかに記載のプログラムデバッグ装置。
【請求項５】　プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応づけるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルに所定のステップ毎に翻訳し実行するインタプリタと、
デバッグ時に上記デバッグスクリプトを有効にするか否かを指定するデバッグ指定手段と、
このデバッグ指定手段により上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定する判定手段と、
この判定手段により上記デバッグスクリプトにプログラムの制御が到達したことを判定したときに、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行するプログラムデバッグ装置。
【請求項６】　上記インタプリタは、上記ソースファイルを翻訳する際所定の記述によりデバッグスクリプトを識別して翻訳し実行することを特徴とする請求項５記載のプログラムデバッグ装置。
【請求項７】　上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記インタプリタは、上記ソースファイルを翻訳する際に上記識別子にて上記デバッグスクリプトを識別して翻訳し実行することを特徴とする請求項５記載のプログラムデバッグ装置。
【請求項８】　上記デバッグ指定手段は、上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、
上記判定手段は、上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項５乃至７のいずれかに記載のプログラムデバッグ装置。
【請求項９】　プログラムの実行時にログを出力するコードであるログスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記ログスクリプト部分であるログスクリプトファイルと、上記ソースファイルのソース行の位置と上記ログスクリプトを対応づけるログスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルするコンパイラと、
デバッグ時に上記ログスクリプトを有効にするか否かを指定するデバッグ指定手段と、
このデバッグ指定手段により上記ログスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記ログスクリプトにプログラムの制御が到達したことを判定する判定手段と、
この判定手段により上記ログスクリプトにプログラムの制御が到達したことを判定した時に、上記ログスクリプト起動情報に基づいて対応した上記ログスクリプトファイルを実行するプログラムデバッグ装置。
【請求項１０】　上記コンパイラは、上記ソースファイ

ルをコンパイルする際所定の記述によりログスクリプトを識別しコンパイルすることを特徴とする請求項９記載のプログラムデバッグ装置。

【請求項１１】　上記ソースファイルは、上記ログスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、
上記コンパイラは、上記ソースファイルをコンパイルする際に上記識別子にて上記ログスクリプトを識別しコンパイルすることを特徴とする請求項９記載のプログラムデバッグ装置。

【請求項１２】　上記デバッグ指定手段は、上記ログスクリプトを有効にした場合、上記ログスクリプトを識別できる命令を上記ログスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、
上記判定手段は、上記命令に基づき上記ログスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項９乃至１１のいずれかに記載のプログラムデバッグ装置。

【請求項１３】　プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応づけるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、
デバッグ時に、上記デバッグスクリプトを有効にするか否かを指定し、
上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、
上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とするプログラムデバッグ方法。

【請求項１４】　上記ソースファイルをコンパイルする際に所定の記述により上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項１３記載のプログラムデバッグ方法。

【請求項１５】　上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、
上記ソースファイルをコンパイルする際に上記識別子にて上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項１３記載のプログラムデバッグ方法。

【請求項１６】　上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、
上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項１３乃至１５のいずれかに記載のプログラムデバッグ方法。

【請求項１７】　プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルに所定のステップ毎に翻訳し実行し、
デバッグ時に上記デバッグスクリプトを有効にするか否かを指定し、
上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、
上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とするプログラムデバッグ方法。

【請求項１８】　上記ソースファイルを翻訳する際に所定の記述によりデバッグスクリプトを識別して翻訳し実行することを特徴とする請求項１７記載のプログラムデバッグ方法。

【請求項１９】　上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、
上記ソースファイルを翻訳する際に上記識別子にて上記デバッグスクリプトを識別して翻訳し実行することを特徴とする請求項１７記載のプログラムデバッグ方法。

【請求項２０】　上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、
上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項１７乃至１９のいずれかに記載のプログラムデバッグ方法。

【請求項２１】　プログラムの実行時にログを出力するコードであるログスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記ログスクリプト部分であるログスクリプトファイルと、上記ソースファイルのソース行の位置と上記ログスクリプトを対応付けるログスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、

デバッグ時に上記ログスクリプトを有効にするか否かを指定し、
上記ログスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記ログスクリプトにプログラムの制御が到達したことを判定し、
上記ログスクリプトにプログラムの制御が到達したことを判定した際に、上記ログスクリプト起動情報に基づいて対応した上記ログスクリプトファイルを実行するプログラムデバッグ方法。
【請求項２２】　上記ソースファイルをコンパイルする際所定の記述によりログスクリプトを識別しコンパイルすることを特徴とする請求項２１記載のプログラムデバッグ方法。
【請求項２３】　上記ソースファイルは、上記ログスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、
上記ソースファイルをコンパイルする際に上記識別子にて上記ログスクリプトを識別しコンパイルすることを特徴とする請求項２１記載のプログラムデバッグ方法。
【請求項２４】　上記ログスクリプトを有効にした場合、上記ログスクリプトを識別できる命令を上記ログスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、
上記命令に基づき上記ログスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項２１乃至２３のいずれかに記載のプログラムデバッグ方法。
【請求項２５】　プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、
デバッグ時に上記デバッグスクリプトを有効にするか否かを指定し、
上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、
上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行するプログラムデバッグ方法のプログラム情報を格納したコンピュータ読み取り可能な記録媒体。
【請求項２６】　プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルに所定のステップ毎に翻訳し実行して、
デバッグ時に上記デバッグスクリプトを有効にするか否かを指定し、
上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、
上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行するプログラムデバッグ方法のプログラム情報を格納したコンピュータ読み取り可能な記録媒体。
【請求項２７】　プログラムの実行時にログを出力するコードであるログスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記ログスクリプト部分であるログスクリプトファイルと、上記ソースファイルのソース行の位置と上記ログスクリプトを対応づけるログスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、
デバッグ時に上記ログスクリプトを有効にするか否かを指定し、
上記ログスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際上記ログスクリプトにプログラムの制御が到達したことを判定し、
上記ログスクリプトにプログラムの制御が到達したことを判定した際に、上記ログスクリプト起動情報に基づいて対応した上記ログスクリプトファイルを実行するプログラムデバッグ方法のプログラム情報を格納したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、コンピュータプログラム開発時のコンパイラと、デバッガを備えたプログラムデバッグ装置、プログラムデバッグ方法、及びその方法を記録したコンピュータ読取り可能な記録媒体に関する。
【０００２】
【従来の技術】従来、組込みシステムのアプリケーション開発の際等に於いて、ターゲットシステム上のプログラムのデバッグを行なう際、エラーチェックルーチンを組込んでデバッグを行ない、プログラムをリリースする際にはエラーチェックルーチンを含まないコードを改めて生成するケースが多々あった。
【０００３】この場合は、再コンパイルが必要であり開発効率が悪い。また、エラーチェックルーチンの有無というデバッグ時とリリース時ではプログラム自体が異なることによる物件管理面での問題、エラーチェックルー

チンを含むためデバッグ時により多くのメモリを必要とするという問題もあった。

【０００４】一般に、プログラム開発者は、プログラムを記述する場合、それをデバッグすることを意識して作成する場合が多い。具体的には、ソースファイルにエラーチェックルーチンを含めておくことが多い。そしてコンパイルオプションでこれを有効にしてコンパイルし、実行モジュールを作成してデバッグする。

【０００５】デバッグが完了すると、このようなコードを無効にしてコンパイルすることで最終プログラムを生成する。特に、組込み製品でのプログラムでは最終的にエラー処理ルーチンは除きたい場合が多々ある。

【０００６】また、多くのデバッガでは、スクリプト記述が可能であり、新たなデバッグコマンドを記述したりすることはできるが、ソースプログラムと独立である。具体的なイメージを擬似言語で記述すれば、以下のようなコードを開発者が記述することになる。

【０００７】

```
Function F () {
.....
#ifdef DEBUG
   error check code ...
#endif
....
#ifdef DEBUG
   error check code ...
#endif
....
}
....
```

ＤＥＢＵＧをオンにして（コンパイルして）オブジェクトを生成しデバッグ完了したあと、ＤＥＢＵＧをオフにして再度（コンパイルして）オブジェクトを生成し最終物件を作成する。

【０００８】これでは、デバッグ時のオブジェクトと最終的なオブジェクトとが別ものになり、管理上の問題を生じたり、再コンパイルが必要で開発効率が悪い。また、デバッグ時では、最終プログラムより多くのコードを含むため、一時的にせよ、メモリを多く必要としてしまう。これは、メモリに厳しい制限のある組込み製品などの場合、問題となる場合がある。

【０００９】

【発明が解決しようとする課題】上述したように問題点を解決するには、以下のような開発環境が提供されることが望ましい。１．デバッグ時物件と、最終物件とを同じ実行オブジェクトで管理できる。

【００１０】２．デバッグ処理時に、ターゲットマシン側のメモリ負担がないようにする。これらの開発環境は、コンパイラやデバッガに係わる部分を従来と変更することで実現される。

【００１１】そこで、本発明は上記事情を考慮してなされたもので、デバッグ用スクリプト記述やプログラムの記述言語による（主にエラーチェックなどの）記述を、ソースファイル内に記述可能で、コンパイラが自動的にそれを抽出してオブジェクトファイル内に保存し、デバッグ時の開発者（デバッガ操作者）の要求に従い、プログラム実行時に自動的にデバッグ用スクリプト記述やプログラムの記述言語による（主にエラーチェックなどの）記述を実行することで、デバッグ時物件と最終物件とを同一のオブジェクトファイルとして扱うことができ、且つクロス開発の場合に、デバッグのためのコードをターゲットマシン内にロードすることを回避避することができるプログラムデバッグ装置、及びプログラム管理方法、並びにその方法を記録したコンピュータ読取り可能な記録媒体を提供することを目的とする。

【００１２】

【課題を解決するための手段】本発明は、上記目的を達成するため、本発明のプログラムデバッグ装置は、プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルするコンパイラと、デバッグ時に上記デバッグスクリプトを有効にするか否かを指定するデバッグ指定手段と、このデバッグ指定手段により上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際上記デバッグスクリプトにプログラムの制御が到達したことを判定する判定手段と、この判定手段により上記デバッグスクリプトにプログラムの制御が到達したことを判定した時に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とする。

【００１３】この構成によれば、デバッグ時のプログラムと最終プログラムを同一のオブジェクトファイルとして扱うことができる。また、デバッグのためのコードは、別のオブジェクトファイルとして管理するので、デバッグのためのコードをターゲットマシン内にロードすることを避けることができる。さらに、上記構成のコンパイラをインタプリタに置き換えても同様な効果が得られる。そして、デバッグスクリプトをログ出力コードに置き換え、このログ出力コードをデバッグ時に利用しても同様な効果が得られる。

【００１４】

【発明の実施の形態】以下、図面を参照しながら本発明の一実施の形態を説明する。

（第１の実施形態）図１は本発明の実施形態の前提となるシステム構成を示すブロック図である。

【００１５】ここではデバッグ対象となるプログラムが動作するマシンをターゲットマシン１１、プログラムを開発（生成）するマシンをホストマシン１０と呼ぶ。両者は、同一のハードウェアであっても良いが、ここでは以下のようにターゲットマシン１１で動作するプログラムを、別個のホストマシン１０を用いて開発する場合を想定する。また、両者は特に図示しないが、ＣＰＵ、メモリ、ＨＤＤ等の補助記憶装置、表示装置、通信手段等、通常のコンピュータが備える構成を当然に有しているものとする。そして、両者はＬＡＮ（Local Area Network）やシリアル等の通信媒体／手段たる例えばネットワーク１２で接続されており、互いにデータを送受信する。

【００１６】図２を参照して、ホストマシン１０上の構成を説明する。ホストマシン１０では、プログラムソース（ソースファイル）２３をコンパイラ２１でコンパイルする。そして、作成されたオブジェクトファイル２４より、実行イメージをホスト上のデバッガ２２が、ネットワーク１２を介して、ターゲットマシン１１上にダウンロードする。

【００１７】デバッガ２２は、プログラムのシンボル情報を取り込むことで、ユーザはシンボリックなデバッグをホストマシン１０上で行うことができる。ユーザの記述したソースファイル２３は、通常のプログラム以外に「デバッグスクリプト」を含めることができる。これは、デバッグ時に、オプションにより該当する部分にプログラムが到達した場合、そのデバッグスクリプトを実行することを保証するものである。このオプションを指定しなければ、この部分は全く実行されない。

【００１８】本ツールでは、コンパイラが通常のオブジェクトファイル２４内に、デバッグスクリプトを保存するセクションを定義する。これにより、コンパイルしたオブジェクトファイル２４には、プログラムコードに相当するセクション（テキストセクション）、データに相当するセクション（データセクション）等を含む通常のプログラムの実行オブジェクトファイルのセクションと、デバッグスクリプトのオブジェクトファイルを保存するデバッグスクリプトセクションと、デバッグスクリプトとソース行の位置を対応づけるデバッグスクリプト起動情報等を含むシンボルテーブルが含まれる。

【００１９】デバッガ２２は、デバッグスクリプトを利用しない場合、通常のリモートデバッグシステムと同様な様相を呈する。デバッグスクリプトオプションが指定された場合でも、プログラムのターゲットマシン１１へのダウンロード機能については全く同じである。

【００２０】図３を参照して、ターゲットマシン１１の構成を説明する。本実施形態では、通常のターゲットシステムでよく用いられるターゲットモニタ３１を仮定するが必須ではない。ターゲットモニタ３１は、ホストマシン１０との通信機能を備え、デバッグ実行時にブレーク命令により例外が発生した場合ホストマシン１０に通知する。３２は、ホストマシン１０からダウンロードされた対象プログラム（オブジェクトファイル）である。

【００２１】次に、ホスト側の言語処理システムの説明をする。ユーザプログラムは、通常の高級プログラミング言語と、この言語のコンパイラの存在を仮定する。上記コンパイラは、特に図示しないが、構文解析部、最適化部、コード生成部などから構成される。その内、本実施形態で主に関係があるのは、構文解析部である。

【００２２】構文解析部では、デバッグスクリプト部とそうでない部分を切り分けて処理する。この識別は、プログラム開発者がソースファイルを作成する際に、「コンパイラ指示子」をソースファイル内に埋め込むことで行う。デバッグ用スクリプトの具体的実施例としては、「ソースプログラム本体と同じ言語（場合によっては、文法的にサブセット）によって記述」、もしくは「デバッガコマンドからなる記述」、もしくは上記が混在したものが考えられる。

【００２３】Ｃ言語風の表現では、以下のようにソースファイルの記述の中に、コンパイル指示子で、デバッグコマンド部分などを記述する。コンパイラの構文解析部は、この部分を通常のプログラムと別の処理を行う。

【００２４】下記例でコンパイル指示子とは、「＃ｉｆｄｅｆ　ｄｅｂｕｇ＿ｂｙ＿ｃｏｍｍａｎｄ　．．．＃ｅｎｄｉｆ」「＃ｉｆｄｅｆ　ｄｅｂｕｇ＿ｂｙ＿Ｃ　．．．　＃ｅｎｄｉｆ」を指す。これらは、プログラムの本来の記述ではなく、コンパイラするときのコンパイラに対する動作指示となる。

【００２５】この例では、コンパイラは、「＃ｉｆｄｅｆ　ｄｅｂｕｇ＿ｂｙ＿ｃｏｍｍａｎｄ」を発見すると、ここからは普通のプログラムではなく、デバッグコマンドであると認識して処理する。また、同様に「＃ｉｆｄｅｆ　ｄｅｂｕｇ＿ｂｙ＿Ｃ」を発見すると、ここからはデバッグコマンドに翻訳すると認識する。

```
func(int a, int b){
int c;
#ifdef debug_by_command
  watch a
print a
#endif
#ifdef debug_by_C

//ここからデバッガコマンド
//  デバッガコマンドの例


//Ｃ言語のチェック
//  →デバッグコマンドに変換する
```

```
 c= convert (a, b);                    //  →関数の一時実行コマンドに変換
                                       //  call $1 = check(a, b)
                                       // set c = $1
if (a > b) printf ("error:a = %d¥n ", a); //  →式の評価に変換：
                                       // set $2 = a
                                       // set $3 = b
                                       // if ($2 > $3) call printf
                                       //("error:a = %d¥n ", a)
#endif
c = ....
....
}
```

上記のように「コンパイラ指示子」では、その記述言語の種類を指定する。今、その種類として、ソースと同一のプログラム言語、デバッガコマンドを想定する。ソースと同一のプログラム言語の場合、これを自動的にデバッガコマンドに変換する。

【００２６】図４を参照して、ホストマシン１０上での上記の様なソースファイルのコンパイル処理の流れを説明する。ソースファイル２３をコンパイラ２１にてコンパイルする場合、まず、コンパイラ２１はコンパイラ指示子が言語記述か否かを判定する（ステップＡ１）。

【００２７】言語記述であれば（ステップＡ１のｙｅｓ）、デバッグスクリプトへの変換ルーチンを呼び出し（ステップＡ４）、デバッグスクリプトセクションを定義して翻訳する（ステップＡ５）。もし、デバッグスプリクトへ変換できなかった場合はエラー処理（ステップＡ６）として終了する。

【００２８】言語記述でなければ（ステップＡ１のｎｏ）、次にコンパイラ指示子がデバッグコマンドか否かを判定する（ステップＡ２）。コンパイラ指示子がデバッグコマンドである場合（ステップＡ２のｙｅｓ）、デバッグスクリプトセクションを定義して翻訳する（ステップＡ５）。コンパイラ指示子がデバッグコマンドでない場合（ステップＡ２のｎｏ）、従来の処理と同様となる（ステップＡ３）。

【００２９】図５に、ホストマシン１０上でのソースファイル２３のコンパイル処理によるオブジェクトファイル生成の概念図を示す。ソースファイル２３の一例であるソースファイル５０をコンパイラ２１により、コンパイルして生成されるオブジェクトファイルは、実行オブジェクトファイル５１、デバッグスクリプトファイル５２、シンボルテーブル５３の３つの部分に分けられる。

【００３０】実行オブジェクトファイル５１は、デバッグスクリプト部を含まない実行オブジェクトファイルのセクションである。デバッグスクリプトファイル５２は、デバッグスクリプトのオブジェクトファイルであるデバッガ用のスクリプトファイルセクションであり、デバッグ時に使用される。シンボルテーブル５３は、シンボル情報を含むファイルのセクションである。シンボル情報とは、例えば、デバッガ用スクリプトを起動する際に用いるソース行とそれに対応する実行オブジェクトファイルのロケーション（位置）とのマッピング情報等である。

【００３１】そして、オブジェクトファイル５１である当該セクションのみがターゲットマシン１１にロードされる。次に、ホストマシン１０上でのデバッグ時の動作説明をする。

【００３２】まず、図６により、通常のブレークポイント指定処理の流れを説明する。ブレークポイントとは、デバッグ時にデバッグ操作者が、必要に応じて実行オブジェクトファイルの実行を中断するためのポイントであり、このブレークポイントを示すため上記実行オブジェクトファイルにはブレーク命令が挿入される。また、この例でのデバッグは、上記ソースファイル５０をコンパイルして生成された実行オブジェクトファイル５１について行うこととし、その際、デバッグスクリプトファイル５２、シンボルテーブル５３を参照する。

【００３３】ブレークポイントの指定は、デバッグ操作者がブレークポイントに対応するソース行を指定して行う（ステップＢ１）。ホストのデバッガ２２が、シンボルテーブル５３にあるシンボル情報（ここでは、ソース行とそれに対応する実行オブジェクトファイル５１のロケーションとのマッピング情報）に基づき、上記指定したブレークポイントに対する実行オブジェクトファイル５１のロケーションにブレーク命令を書き込む（ステップＢ２）。その際に、上記ステップＢ１で指定したブレーク命令を書き込んだ実行オブジェクトファイル５１のロケーションに有った元々の命令はデバッガ２２のメモリの中に保存しておく（ステップＢ３）。

【００３４】次に、図７により、通常のブレークポイントの解除処理の流れを説明する。デバッガ操作者が、ブレークポイントを解除する（ステップＣ１）。デバッガ２２は、実行オブジェクトファイル５１の解除されたブレークポイントのブレーク命令のロケーションに、メモリに保存しておいたこのブレーク命令のロケーションに有った元々の命令を書き込む（ステップＣ２）。これにより、このロケーションの命令を実行してもブレーク命

令による例外は発生しない。

【００３５】本実施形態では、デバッグ時にターゲットマシン１１で実行オブジェクトファイル５１を実行する際に、実行オブジェクトファイル５１に挿入したブレーク命令にプログラムの処理が到達した時、このブレーク命令がデバッグスクリプトに対応している場合、ホストマシン１０上のシンボルテーブル５３のシンボル情報に基づき、ホストマシン１０上のデバッグスクリプトファイル５２からこのブレークポイントに対応したデバッグスクリプトを実行して、デバッグを行う。

【００３６】デバッグ操作者によるデバッグスクリプト有効化または無効化の指示について以下に説明する。まず、図８により、デバッグスクリプト有効化処理の流れを説明する。デバッグ操作者が、デバッグスクリプト有効化を指定する（ステップＤ１）。デバッグスクリプト有効化が指定されると、デバッガ２２は、シンボルテーブル５３より、各デバッグスクリプト毎に、実行オブジェクトファイル５１の該当するロケーションを調べそこにブレーク命令を書き込む（ステップＤ２）。ブレーク命令のパラメータとして、各デバッグスクリプト毎のＩＤを示すデバッグスクリプトＩＤを挿入しておく。ステップＤ２の際に、実行オブジェクトファイル５１の該当するロケーションにあった元々の命令をデバッガのメモリに保存しておく（ステップＤ３）。

【００３７】続いて図９により、デバッグスクリプト無効化処理の流れを説明する。デバッグ操作者が、デバッグスクリプト無効化を指定する（ステップＥ１）。デバッグスクリプト無効化が指定されると、デバッガ２２は、シンボルテーブル５３より、各デバッグスクリプト毎に、実行オブジェクトファイル５１の該当するロケーションを調べ、対応するブレーク命令のロケーションにデバッガの保存領域にコピーしてあったそのロケーションに有った元々の命令を書き込む（ステップＥ２）。これにより、デバッグ時に、ブレーク命令によるブレーク例外は発生せずデバッグスクリプトは無視される。

【００３８】図１０により、デバッグ時の処理の流れを説明する。デバッグ時、ターゲットマシン１１で実行オブジェクトファイル５１を実行する際、ブレーク命令にプログラムが到達すると（ステップＦ１）と、通常のＣＰＵでは例外が発生する。例外とは、外部からの割込み、ゼロによる割り算などのときにも起こるもので、これが発生するとＣＰＵは、現在のプログラムの実行を止めて、図示しない例外処理ハンドラにジャンプする。

【００３９】例外処理ハンドラでは、どのような例外が発生してここに飛んできたか、ＣＰＵのレジスタなどを参照して判定し、それぞれサブの処理ルーチンに飛ばす。例外が発生した時、それが「ブレーク命令」によって起こったものだとわかると、ターゲットモニタ３１にそれを知らせる。ターゲットモニタ３１は、内蔵する通信機能によりホストマシン１０のデバッガ２２に、ブレークが起きたことを知らせる。

【００４０】ホストマシン１０のデバッガ２２は、ブレークポイントを張っていることを分かっているので、デバッグ操作者にそれをわかるようにターゲットマシン１１の表示装置に表示する。

【００４１】ターゲットモニタ３１は、次のデバッガ２２からのコマンドを待つ。ここでは、まずホストマシン１０上のデバッガ２２は、発生した例外がデバッグ例外か否かを判定する（ステップＦ２）。

【００４２】デバッグ例外でない場合（ステップＦ２のｎｏ）、通常の例外処理を実行し（ステップＦ６）、終了する。デバッグ例外で有る場合（ステップＦ２のｙｅｓ）、デバッグスクリプト対応か否か判定する（ステップＦ３）。

【００４３】デバッグスクリプト対応でない場合（ステップＦ３のｎｏ）、通常のデバッグ例外処理を実行し（ステップＦ７）、終了する。デバッグスクリプト対応である場合（ステップＦ３のｙｅｓ）、デバッガ２２は、デバッグスクリプトＩＤを調べ、シンボルテーブル５３のシンボル情報に基づきホストマシン１０上のデバッグスクリプトファイル５２から各デバッガコマンドを実行し（ステップＦ４）、例外処理より復帰する（ステップＦ５）。例外処理より復帰したら、続けてプログラムを実行する。

【００４４】以上のようにすることで、デバッグ時プログラムと最終プログラムを同一のオブジェクトファイルとして扱うことができ、またデバッグのためのコードをターゲットマシン内にロードすることを避けることができる。

【００４５】上述の第１の実施形態では、コンパイラを仮定していたが、インタプリタ方式でも同様に扱うことができる。また、プログラム言語としては、複数種の存在を仮定することもできる。さらに、コンパイル指示子の種類により、記述の種類を判別したが、より高度にコンパイラでの自動判定することが考えられる。

【００４６】所定の記述、例えば、関数名、変数名がｄｅｂｕｇ＿．．．を含む式や、それを含む文を判定しデバッグコマンドに自動変換するなど。例として、以下のようなものがあげられる。

```
func (int a, int b){
int c;

debug_watch a                    //デバッグコマンドとして処理： watch a
debug_print a                    //デバッグコマンドとして処理： print a
```

```
printf ("%d¥n", c);          //通常プログラムとして処理
c = debug_convert (a, b);    //デバッグコマンドとして処理：
                             // call $1 = check(a, b)
                             // set c = $1

}
```

（第２の実施形態）次に、本発明に係わる第２の実施形態について説明する。

【００４７】本実施形態は、上記第１の実施形態でデバッグスクリプトに相当する部分を、デバッグ時にログとして出力されるコードに置き換えたものと考えることができる。

【００４８】本実施形態でのソースファイル例であるソースファイル７０をコンパイルしてオブジェクトファイルを生成する概念図を図１１に示す。ソースファイル７０をコンパイラにてコンパイルすると、生成されるオブジェクトファイルは、実行オブジェクトファイル７１、ログスクリプトファイルセクション７２、シンボルテーブル７３の３つの部分に分けられる。

【００４９】実行オブジェクトファイル７１は、デバッグスクリプト部（この場合ログとして出力されるコード）を含まない実行オブジェクトファイルのセクションである。ログスクリプトファイルセクション７２は、デバッガ用のスクリプトファイルセクションであり、ログ出力がデバッグ用に使用される。

【００５０】シンボルテーブル７３は、シンボル情報を含むファイルのセクションである。シンボル情報とは、例えばデバッガ用スクリプトであるログを起動する際に用いるソース行とそれに対応する実行オブジェクトファイルのロケーションとのマッピング情報等である。

【００５１】本実施形態のログ出力の有効化、無効化処理は、第１の実施形態のデバッグスクリプトの有効化、無効化処理と同様に扱うことができる。ログ出力が有効化されれば、ログ出力命令のブレークポイントにプログラムが到達した場合、ログスクリプトファイルセクション７２からこのログ出力命令に対応するログ出力を選択して実行しデバッグを行うことができる。

【００５２】

【発明の効果】以上詳記したように本発明によれば、デバッグ時物件と最終物件とを同一のオブジェクトファイルとして扱うことができ、デバッグのためのコードをターゲットマシン内にロードすることを回避することで、ターゲットマシンのメモリ要求量を抑制することができる。また、ソースファイルとデバッグ用スクリプトを一括して管理できる。そして、デバッグスクリプトとして、プログラムを記述している言語そのものが使用でき、プログラマ（プログラム開発者）からみると、それがデバッグコマンドに翻訳されて実行されるということを意識しないで良いという優れた効果を奏する。

【図面の簡単な説明】

【図１】本発明の第１の実施形態に於ける装置の概略構成を示す図。

【図２】同実施の形態に係るホストマシン上の構成要素を示す概念図。

【図３】同実施の形態に係るターゲットマシン上の構成要素を示す概念図。

【図４】同実施の形態に係るソースファイルのコンパイル処理の流れを示すフローチャート。

【図５】同実施の形態に係るソースファイルをコンパイルして生成されるオブジェクトフアイルの概念図。

【図６】同実施の形態に係る通常のブレークポイント指定処理の流れを示すフローチャート。

【図７】同実施の形態に係る通常のブレークポイント解除処理の流れを示すフローチャート。

【図８】同実施の形態に係るデバッグスクリプト有効化処理の流れを示すフローチャート。

【図９】同実施の形態に係るデバッグスクリプト無効化処理の流れを示すフローチャート。

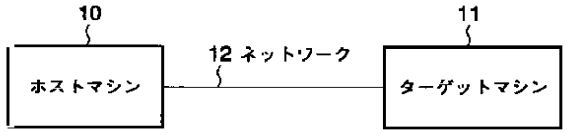
【図１０】同実施の形態に係るデバッグ時の処理の流れを示すフローチャート。

【図１１】本発明の第２の実施形態に於ける、ソースファイルをコンパイルして生成されるオブジェクトファイルの概念図。

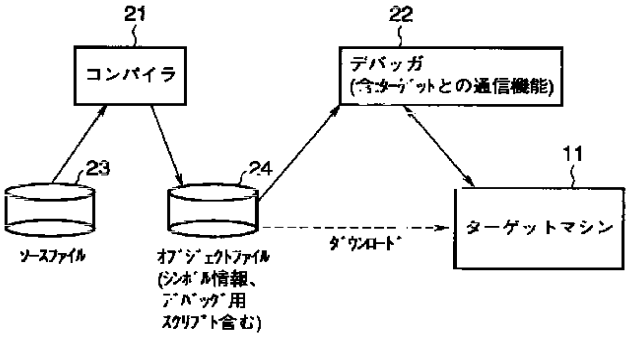
【符号の説明】

１０…ホストマシン
１１…ターゲットマシン
１２…ネットワーク
２１…コンパイラ
２２…デバッガ
２３…ソースファイル
２４…オブジェクトファイル
３１…ターゲットモニタ
３２…ダウンロードされた対象プログラム

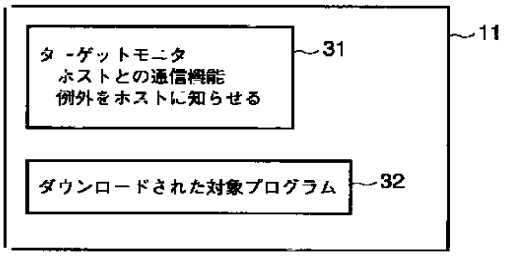
【図１】



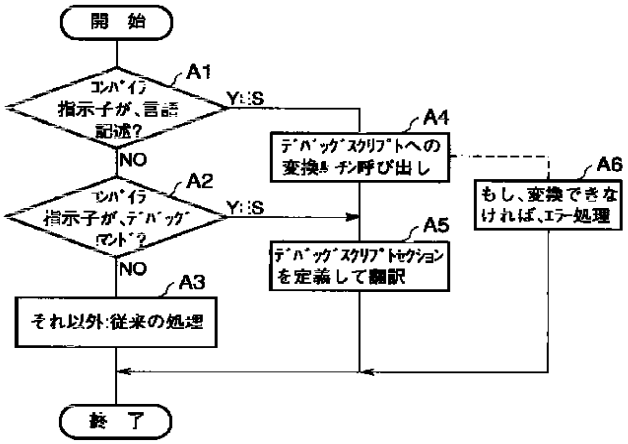
【図２】



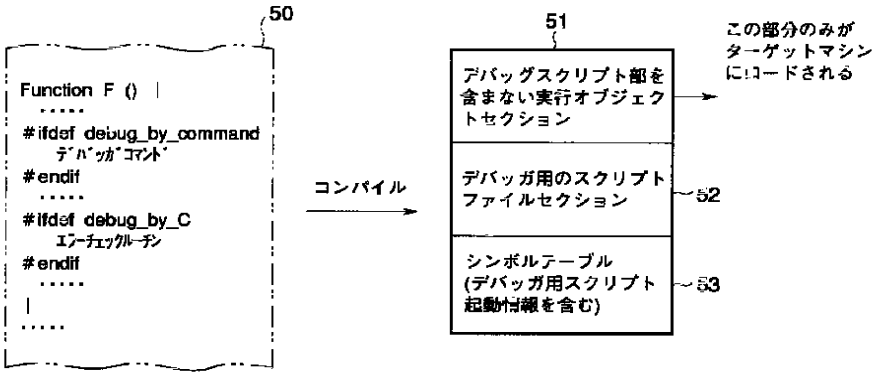
【図３】



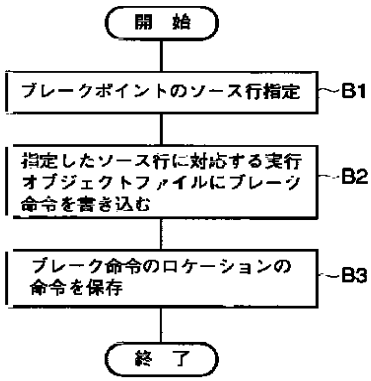
【図４】



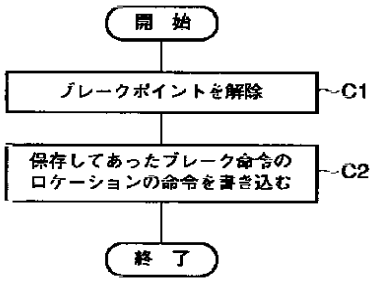
【図５】



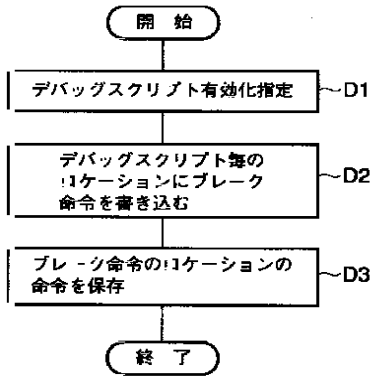
【図６】



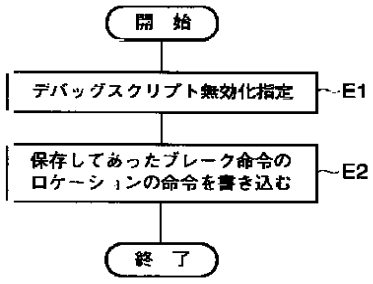
【図７】



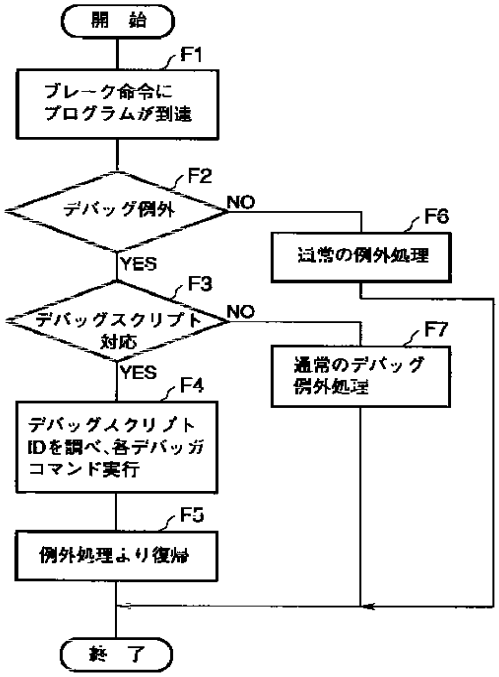
【図８】



【図９】



【図１０】



【図１１】